ANALYSIS ON AN IMPLEMENTATION OF THE GENS-DOMINGOS SUM-PRODUCT NETWORK STRUCTURAL LEARNING SCHEMA

Renato Lui Geh
Computer Science
Institute of Mathematics and Statistics
University of São Paulo
`renatolg@ime.usp.br`

Abstract. Sum-Product Networks (SPNs) are a class of deep probabilistic graphical models. Inference in them is linear in the number of edges of the graph. Furthermore, exact inference is achieved, in a valid SPN, by running through its edges twice at most, making exact inference linear. The Gens-Domingos SPN Schema is an algorithm for structural learning on such models. In this paper we present an implementation of such schema, analyzing its complexity, discoursing implementational and theoretical details, and finally presenting results and experiments achieved with this implementation.

Keywords `cluster analysis; data mining; probabilistic graphical models; tractable models; machine learning; deep learning`

1. Introduction

A Sum-Product Network (SPN) is a probabilistic graphical model that represents a tractable distribution of probability. If an SPN is valid, then we can perform exact inference in time linear to the graph's edges. Its syntax is different to other conventional models (read bayesian and markov networks) in the sense that its graph does not explicitly model events and (in) dependencies between variables. That is, whilst variables in a bayesian network are represented as nodes in the graph, with each edge connecting two nodes asserting a dependency relationship between the connected variables, a node in an SPN may not necessarily represent a variable or event, neither an edge connecting two nodes represent dependence. In this sense, SPNs can be seen as a type of probabilistic Artificial Neural Network (ANN). However, whilst neural networks represent a function, SPNs model a tractable probability distribution. Furthermore, SPNs are distinct from standard neural networks seeing that, whereas ANNs have only one type of neuron with an activation function mapping to values in $[0, 1]$, SPNs have two kind of neurons, which we will see in the next sections. Still, SPNs retain certain important characteristics from ANNs as we will discuss later, with mainly its deep structure properties [DB11] as the most interesting feature.

The Gens-Domingos Schema [GD13], or `LearnGD` as we will reference it throughout this paper, is an SPN structural learning algorithm proposed by Robert Gens and Pedro Domingos. Gens and Domingos call it a schema because it only provides a template of what the algorithm should be like. We will discuss `LearnGD` in details in the next section. This paper documents a particular implementation of the GD schema. Other implementations may have different results.

In this document, we show how we implemented the `LearnGD` algorithm. We analyse the complexity of each algorithm component in detail, later referring to such analyses when drawing conclusions on the overall complexity of the algorithm. As we have mentioned before, since the `LearnGD` schema depends heavily on implementation, the complexity we achieve in this particular case may differ from other implementations. After each analysis, we then look at the algorithm as whole, drawing conclusions on time and memory usage, as well as implementation details that could potentially decrease the algorithm runtime. We also comment on how to implement better concurrency then how it is currently coded in our implementation. We then show some results on experiments made on image classification and image completion.

2. Sum-Product Networks

In this section we will define SPNs differently from other articles [GD13; PD11; DV12] as the original more convoluted definition is of little use for the `LearnGD` algorithm. Our definition is almost identical to the original `LearnGD` article [GD13], with the exception that we assume that an SPN is already normalized. This fact changes nothing, since Peharz *et al* recently proved that normalized SPNs have as much representability power as unnormalized SPNs [Peh+15]. Before we enunciate the formal definition of an SPN, we will give an informal, vague definition of an SPN in order to explain what completeness, consistency, validity and decomposability — which are an important set of definitions — of an SPN mean.

A sum-product network represents a tractable probability distribution through a DAG. Such digraph must always be weakly connected. A node can either be a leaf, a sum, or a product node. The scope of a node is the set of all variables present in all its descendants. Leaf nodes are tractable probability distributions and their scope is the scope of its distribution, sum nodes represent the summing out of the variables in its scope and product nodes act as feature hierarchy. An edge that has its origin from a sum node has a non-negative weight. We refer to a sub-SPN S rooted at node i as $S(i)$, while the SPN rooted at its root is denoted as $S(\cdot)$ or simply S. The scope of a node will be denoted as $\mathrm{Sc}(i)$, where i is a node. The set of children of a node will be denoted as $\mathrm{Ch}(i)$. Similarly, $\mathrm{Pa}(i)$ is the set of parents of node i.

Definition 2.1 (Normalized).
Let S be an SPN and $\Sigma(S)$ be the set of all sum nodes of S. S is normalized iff, for all $\sigma \in \Sigma(S)$, $\sum_{c \in Ch(\sigma)} w_{\sigma c} = 1$ and $0 \leq w_{\sigma c} \leq 1$, where $w_{\sigma c}$ is the weight from edge $\sigma \to c$.

Definition 2.2 (Completeness).
Let S be an SPN and $\Sigma(S)$ be the set of all sum nodes of S. S is complete iff, for all $\sigma \in \Sigma(S)$, $Sc(i) = Sc(j), i \neq j; \forall i, j \in Ch(\sigma)$.

Definition 2.3 (Consistency).
Let S be an SPN, $\Pi(S)$ be the set of all product nodes of S and X a variable in $Sc(S)$. S is consistent iff X takes the same value for all elements in $\Pi(S)$ that contain X.

Definition 2.4 (Validity).
An SPN S is valid iff it always computes the correct probability of evidence S represents.

Theorem 2.1. *An SPN S is valid if it is both complete and consistent.*

Validity guarantees that the SPN will compute not only the correct probability of evidence, but also in time linear to its graph's edges. Therefore, it is preferable to learn valid SPNs. Notice that Theorem 2.1 is not restricted by completeness and consistency. In fact, incomplete and/or inconsistent SPNs can compute the probability of evidence correctly, but consistency and completeness guarantee that all sub-SPNs are also valid.

Definition 2.5 (Decomposability).
Let S be an SPN and $\Pi(S)$ be the set of all product nodes in S. S is decomposable iff, for all $\pi \in \Pi(S)$, $Sc(i) \cap Sc(j) = \emptyset, i \neq j; \forall i, j \in Ch(\pi)$.

It is clear that decomposability implies consistency, therefore if an SPN is both complete and decomposable, than it is also valid. We choose to work with decomposability because it is easier to learn decomposable SPNs then it is to learn consistent ones. We do not lose representation power because a complete and consistent SPN can be transformed into a complete and decomposable SPN in no more than a polynomial number of edge and node additions [Peh+15]. We can now formally define an SPN.

Definition 2.6 (Sum-product network).
A sum-product network (SPN) is a weakly connected DAG that can be recursively defined as following.

An SPN:

*(1) with a single node is a univariate tractable probability distribution (**leaf**);*
*(2) is a normalized weighted sum of SPNs of same scope (**sum**);*
*(3) is a product of SPNs with disjoint scopes (**product**).*

The value of an SPN is defined by its type. Let λ, σ and π be a leaf, sum and product respectively. The values of such SPNs are given by $\lambda(\mathbf{x})$, $\sigma(\mathbf{x})$ and $\pi(\mathbf{x})$, where $\mathbf{x}$ is a certain evidence instantiation.

Leaf: *$\lambda(\mathbf{x})$ is the value of the probability distribution at point $\mathbf{x}$.*
Product: $\pi(\mathbf{x}) = \prod_{c \in Ch(\pi)} c(\mathbf{x})$.
Sum: $\sigma(\mathbf{x}) = \sum_{c \in Ch(\sigma)} w_{\sigma c} c(\mathbf{x})$, *with* $\sum_{c \in Ch(\sigma)} w_{\sigma c} = 1$ *and* $0 \leq w_{\sigma c} \leq 1$.

Note that this definition assumes an SPN to be complete, decomposable and normalized. Other definitions in literature may differ from ours, but as we have mentioned before, for our implementation, this definition is convenient for us. Another observation worthy of notice is the value of $\lambda(\mathbf{x})$. Although here we consider $\mathbf{x}$ to be a multivariate instantiation (i.e. a set of — potentially multiple — variable valuations), we had initially defined a leaf to be a univariate distribution. Although it is possible to attribute leaves as multivariate probability distributions [RL14], for our definition we have chosen to keep a leaf's scope a unit set. Therefore, in the case of a leaf's value, $\mathbf{x}$ is a singleton (univariate) variable instantiation.

3. The `LearnGD` Schema

The `LearnGD` schema was proposed by Robert Gens and Pedro Domingos on *Learning the Structure of Sum-Product Networks* [GD13]. In this section we will outline the schema in pseudo-code and analyse a few properties derived from the algorithm.

Algorithm 1 `LearnGD`

```
Input  Set $D$ of instances (data)
Input  Set $V$ of variables (scope)
Output  An SPN representing a probability distribution given by $D$ and $V$
 1: if $|\mathbf{V}| = 1$ then                                ▷ univariate data sample
 2:     return univariate distribution estimated from $T[V]$ (data of $V$)
 3: end if
 4: Take $V$ and find mutually independent subsets $V_i$ of variables
 5: if possible to partition then               ▷ i.e. we have found independent subsets
 6:     return $\prod_i$ LearnGD ($D$,$V_i$)
 7: else                                          ▷ we cannot say there is independence
 8:     Take $D$ and find $D_j$ subsets of similar instances
 9:     if possible to partition then
10:         return $\sum_i \frac{|D_j|}{|D|} \cdot$ LearnGD ($D_j$,$V$)
11:     else                                             ▷ i.e. data is one big cluster
12:         return fully factorized distribution.
13:     end if
14: end if
```

Let us now, for a moment, suppose that SPNs are not necessarily complete, decomposable and normalized. We shall prove a few results derived from SPNs generated by Algorithm 1.

Lemma 3.1. *An SPN S generated by `LearnGD` is complete, decomposable and normalized.*

Proof. Lines 4–6 show that the scope of each child in a product node of S is a partition of the scope of their parent. Therefore, children have pairwise disjoint scopes on line 6, which proves decomposability for this part of the algorithm. In lines 8–10, since we are clustering similar instances, D is being partitioned but we are not changing V in any way. In fact, line 10 shows that we pass V to all other children. That is, all children of sum nodes have the same scope as their parent, which proves completeness. Let $D_1, \ldots, D_n$ be the subsets of similar instances. By the definition of clustering, $D_1 \cup \ldots \cup D_n = D$ and $D_i \cap D_j = \emptyset$, $i \neq j$, $1 \leq i, j \leq n$. Thus it follows that $\sum_{i=1}^{n} \frac{|D_i|}{|D|} = 1$ and thus line 10 always creates complete and normalized sum nodes. Line 12 is a special case where, if we have discovered that D is one big data cluster, we shall create a product node π in which all children of π are leaves and

$$\bigcup_{\lambda \in \mathrm{Ch}(\pi)} \mathrm{Sc}(\lambda) = \mathrm{Sc}(\pi).$$

In other words, we fully factorize our product node into leaves. In this case, it is obvious that this product node is decomposable. □

`LearnGD` can be divided into three parts:

(1) Is the data univariate? If it is, return a leaf.
(2) Are partitions of the data independent? If they are, return a product node whose children are the independent partitions.
(3) Are partitions of the data similar? If they are, return a sum node whose children are the partition clusters.
(4) In case all else fails, we have a fully factorized distribution.

Going back to our definition of an SPN, we can now take a more intuitive approach and make the following observations:

(1) A leaf is nothing but a local/partitioned/sample distribution of a probability distribution given by a single variable.
(2) A product node determines independence between variables.
(3) A sum node is a clustering of similar data values (i.e. instances that are "alike").

This gives more semantic value to SPNs, whilst still retaining its expressivity. Following this approach, one can easily notice that each "layer" corresponds to a recursive call in `LearnGD`. In fact, each recursive call constructs a hidden layer that tries to partition the SPN even further. This gives SPNs a deep architecture that resembles deep models in that the deeper the model, the more representation power it has [DB11].

Let us now observe the scope of each type of node. A leaf is the trivial case, since it has a single variable in its scope by definition. Each layer above it can have either sum or product nodes. Let us now look at decomposability, that is: if a variable X appears in a child of a product node π, then X cannot appear in another child of π. This gives us the following result:

Proposition 3.1. *Let S be an SPN generated by `LearnGD`, and let $\Lambda(S)$ be the set of all leaves of S. Then, $\forall \lambda \in \Lambda(S)$, we have that, $\forall p \in Pa(\lambda)$, p is a product node.*

Proof. Our proof is by contradiction. Let us assume that $\exists p \in \mathrm{Pa}(\lambda)$ such that p is a sum node and $\exists c^* \in \mathrm{Ch}(p)$ a leaf. From our assumption that p is a sum node, we have that, since the SPN is complete, the scope of all children of p are the same and are all equal to the scope of p. Now let $c \in \mathrm{Ch}(p)$. There must exist another child c such that $c \neq c^*$ because of lines 5 and 9. From that we have $\mathrm{Sc}(c) = \mathrm{Sc}(c^*)$ because of completeness, and since $\mathrm{Sc}(c^*)$ is singular, then c must also be leaf. But it is impossible to have leaves with same scope and same parent (line 1 from Algorithm 1). Therefore, p is actually a product node. $\square$

References

[DB11] Olivier Delalleau and Yoshua Bengio. "Shallow vs. Deep Sum-Product Networks". In: *Advances in Neural Information Processing Systems 24 (NIPS 1011)* (2011).

[DV12] Aaron Dennis and Dan Ventura. "Learning the Architecture of Sum-Product Networks Using Clustering on Variables". In: *Advances in Neural Information Processing Systems* 25 (2012).

[GD13] Robert Gens and Pedro Domingos. "Learning the Structure of Sum-Product Networks". In: *International Conference on Machine Learning* 30 (2013).

[PD11] Hoifung Poon and Pedro Domingos. "Sum-Product Networks: A New Deep Architecture". In: *Uncertainty in Artificial Intelligence* 27 (2011).

[Peh+15] Robert Peharz et al. "On Theoretical Properties of Sum-Product Networks". In: *International Conference on Artificial Intelligence and Statistics 18 (AISTATS 2015)* (2015).

[RL14] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions". In: *International Conference on Machine Learning 31 (ICML 2014)* (2014).